

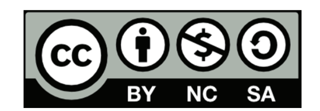
Tema 3: Jerarquía de Memorias

• Conceptos Básicos de Memoria Cache

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya





640K ought to be enough for anybody.

- Bill Gates, 1981

Windows NT addresses 2 Gigabytes of RAM, which is more than any application will ever need.

- Microsoft, on the development of Windows NT, 1992

Índice

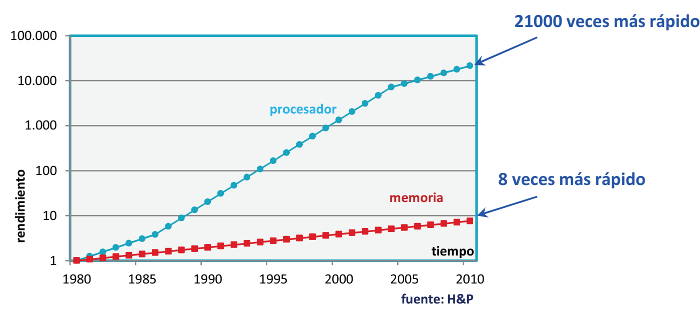
- **Conceptos Básicos Memoria Cache**
  - Introducción
  - Algoritmos de emplazamiento
  - Algoritmos de reemplazo
  - Políticas de escritura
  - Influencia de los parámetros
- Memoria Virtual
- Conceptos Avanzados Memoria Cache
- Memoria Principal
- Conceptos Avanzados Memoria Principal



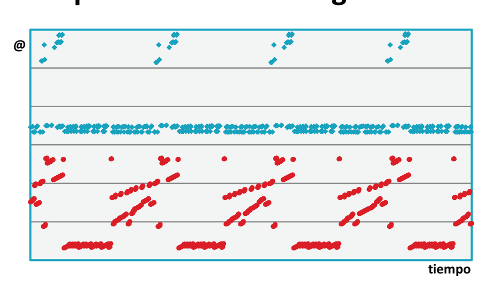
Visión General de la Jerarquía

La jerarquía de Memorias está justificada por las siguientes situaciones:

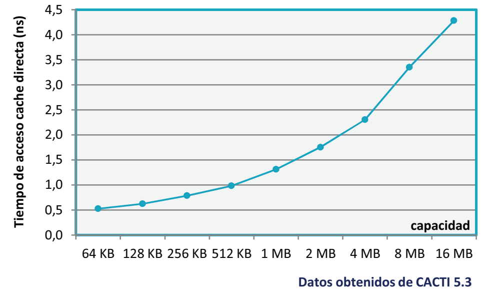
- **Velocidad Procesadores >> Velocidad Memorias.**



- **Propiedades de los Programas.**



- **Propiedades de las memorias.**



Velocidad Procesadores >> Velocidad Memorias

Año	Procesador	Frec.	Número máximo referencias/ciclo	Ancho Banda necesario
1991	DEC Alpha 21064	200 MHz	2.5	2 GB/s
2000	DEC Alpha 21264	600 MHz	5	12 GB/s
2003	Intel Pentium 4	3 GHz	3.75	45 GB/s
2010	Intel Xeon X5680 (6 cores)	3,33 GHz	22.5	300 GB/s

Año	Tipo DRAM (Capacidad)	Frec. Placa Base	Latencia 1r dato / resto	Ancho Banda lect. 32 bytes	Ancho Banda de pico
2000	EDO DRAM (64 Mb)	66 MHz	50 / 20 ns	290.9 MB/s	400 MB/s
2002	SDRAM (128 Mb)	167 MHz	36 / 6 ns	592.6 MB/s	1,33 GB/s
2003	DDR SDRAM (256 Mb)	200 MHz	30 /2.5 ns	853.3 MB/s	3,2 GB/s
2010	DDR3 SDRAM (2 Gb)	1,1 GHz	17 / 0,45 ns	1,74 GB/s	17,6 GB/s

Propiedades de las Memorias

Tipos de Memoria de Semiconductores:

- **Memoria Estática** (SRAM, Static RAM). Cada celda de memoria equivale a 1 biestable (6-8 transistores). En comparación con las DRAM son **rápidas**, tienen un **alto consumo**, **pequeñas** (poca capacidad) y **caras**.

  → **Memoria Cache**

- **Memoria Dinámica** (DRAM, Dynamic RAM). Cada celda se comporta como un condensador (1-1.x transistores). En comparación con las SRAM son **lentas**, tienen un **bajo consumo**, **grandes** (mucha capacidad) y **baratas**. Problema del refresco.

  → **Memoria Principal**

Propiedades de las Memorias

Tipo	Capacidad	Tiempo acceso	Tecnología	Coste por Mbyte
Registros	64 – 1024 bytes	0.3 – 1ns	-	-
Memoria Cache	8Kb – 2 Mb	1 - 5 ns	Semiconductor SRAM	6.75 €
Memoria Principal	1Mb – 1 Gb	10 – 30 ns	Semiconductor DRAM	0.25 €
Memoria Secundaria	10 Gb – 200 Gb	10 – 50 ms	Disco Duro Magnética	0.00125 €

Datos de febrero de 2003



↓Capacidad ⇔ ↓Tiempo de acceso ⇔ ↑Coste por Mbyte



Propiedades de los Programas

- **Regla del 90 / 10**
  - El 90% de todas las referencias a memoria (datos e instrucciones) son realizadas por el 10% del código.
  - Spec2000

%@	% referencias a datos	% referencias a Instrucciones
1%	79,6%	90,7%
2%	84,8%	97,3%
5%	90,4%	99,6%
10%	93,7%	100,0%

El 2% de las posiciones de memoria de datos reciben el 84,8% de todas las referencias a datos.

El 5% de todas las instrucciones reciben el 99,6% de todas las referencias a instrucciones.

Propiedades de los Programas

Localidad Temporal

- Si accedemos a una posición de memoria, es muy probable que se vuelva a acceder a la misma posición en un futuro cercano.

```
X3D(int v[], int w[]) {
  int vert, polig, color;
  …
  for (i=0; i<5000; i++) {
    v[i] = w[i] | 0x01;
    v[i] = v[i] * vert;
    if (v[i] != 0)
      w[i] = polig / v[i];
    v[i] = w[i] + color;
  }
 …
}
```

¿Por qué? Por los bucles.

- **Instrucciones**: dentro de un bucle se accede repetidamente a las mismas instrucciones.
- **Datos**: dentro de un bucle se accede repetidamente a las mismas variables.



Propiedades de los Programas

Localidad Espacial

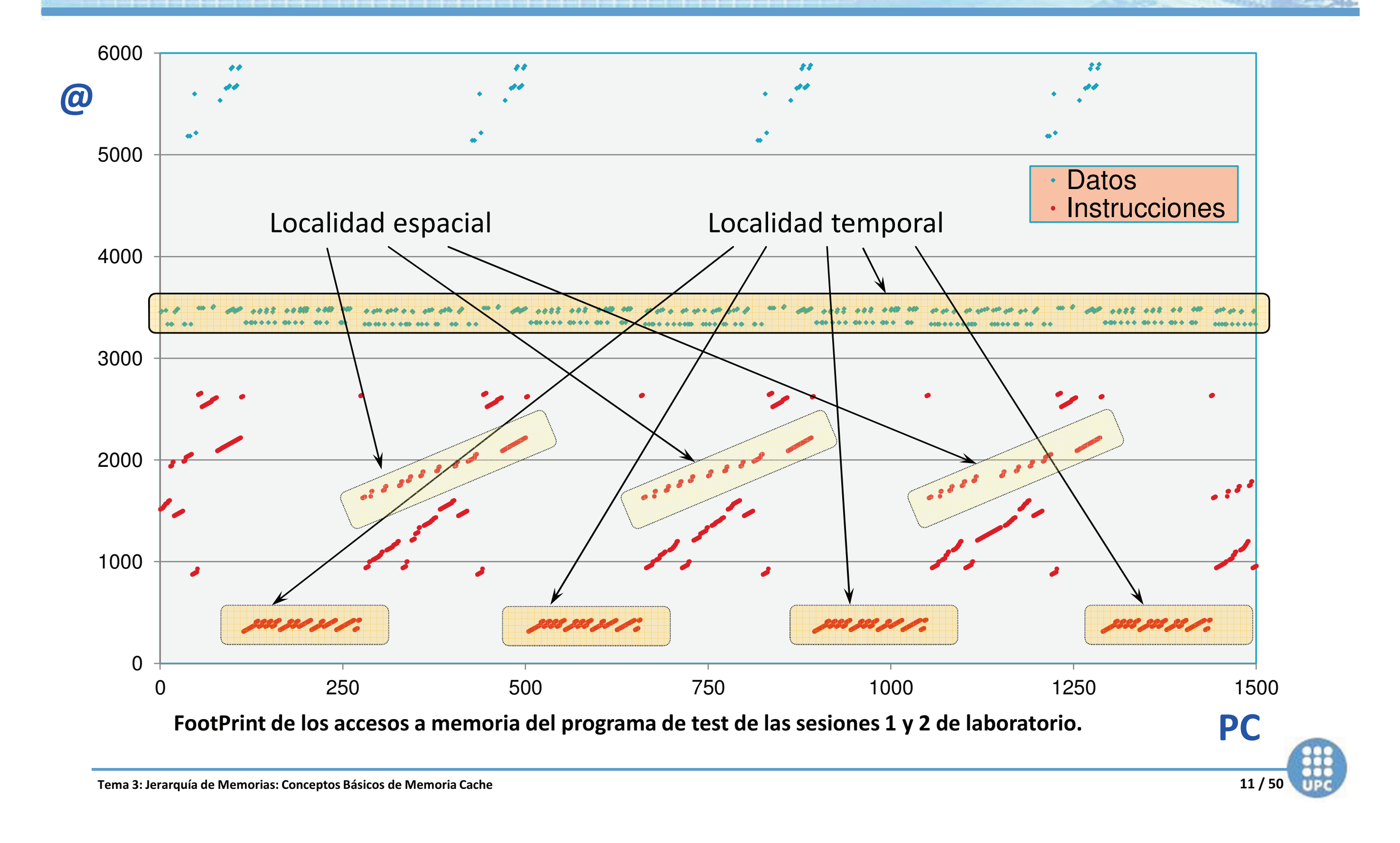
- Si accedemos a una posición de memoria, es muy probable que se acceda a posiciones próximas en un futuro cercano.

```
X3D(int v[], int w[]) {
  int vert, polig, color;
  …
  for (i=0; i<5000; i++) {
    v[i] = w[i] | 0x01;
    v[i] = v[i] * vert;
    if (v[i] != 0)
      w[i] = polig / v[i];
    v[i] = w[i] + color;
  }
 …
}
```

¿Por qué?

- **Instrucciones**: las instrucciones se ejecutan en secuencia.
- **Datos**: los vectores y matrices se suelen recorrer completos en secuencia; los parámetros y variables locales están en el bloque de activación de la subrutina.

Propiedades de los programas

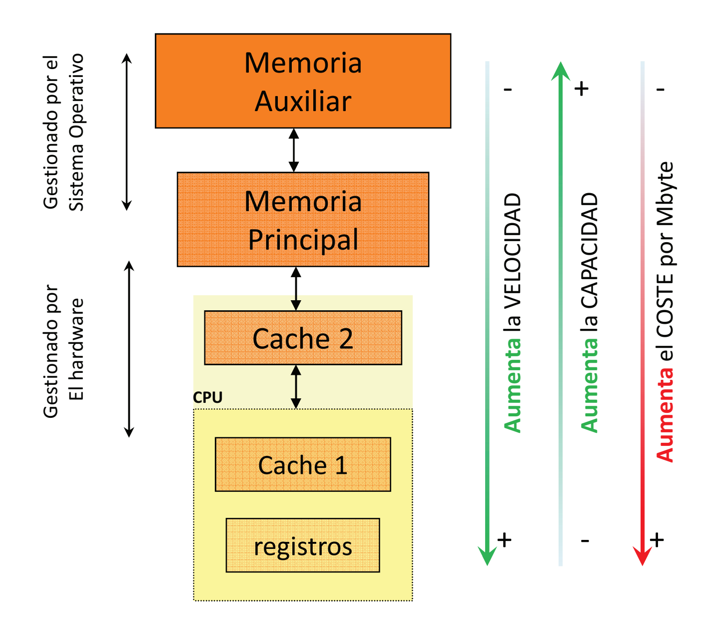


FootPrint de los accesos a memoria del programa de test de las sesiones 1 y 2 de laboratorio.

Propiedades de los Programas

- **¿Cómo podemos aprovechar la localidad?**
  - **Localidad Temporal**: si traemos un dato (o instrucción) de memoria, sería útil guardarlo "cerca" del procesador para que los futuros accesos sean más rápidos.
  - **Localidad Espacial**: si traemos un dato (o instrucción) de memoria, sería útil traer también los datos próximos y dejarlos "cerca" del procesador. Esto sólo tiene sentido si traer datos próximos sólo cuesta un poco más que traer un solo dato.

Solución: Jerarquía de Memorias



Objetivo: que cuando el procesador acceda a un dato, éste se encuentre en los niveles de la jerarquía más cercanos al procesador.

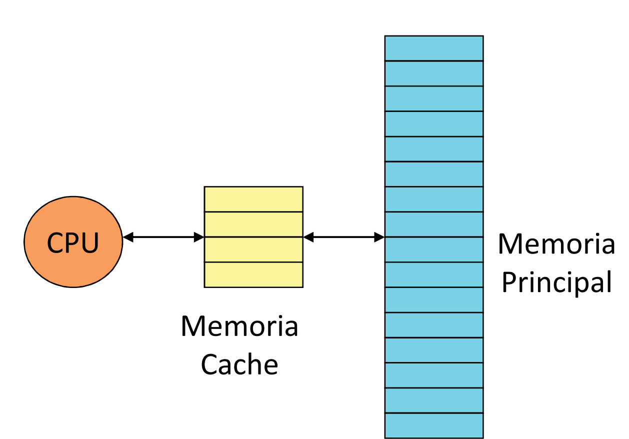
Si se consigue, obtendríamos una memoria con la velocidad de los niveles rápidos, el tamaño de los niveles más grandes y todo ello con un coste razonable.

¡La jerarquía funciona por las propiedades (localidad) de los programas!



Principios de Funcionamiento de las Memorias Cache

- **Memoria Cache**: memoria **pequeña** y **rápida** que almacena una parte del contenido de una memoria más grande y lenta. La memoria cache se encargará de que la información que se almacene sea útil. Esta memoria es transparente al programador.

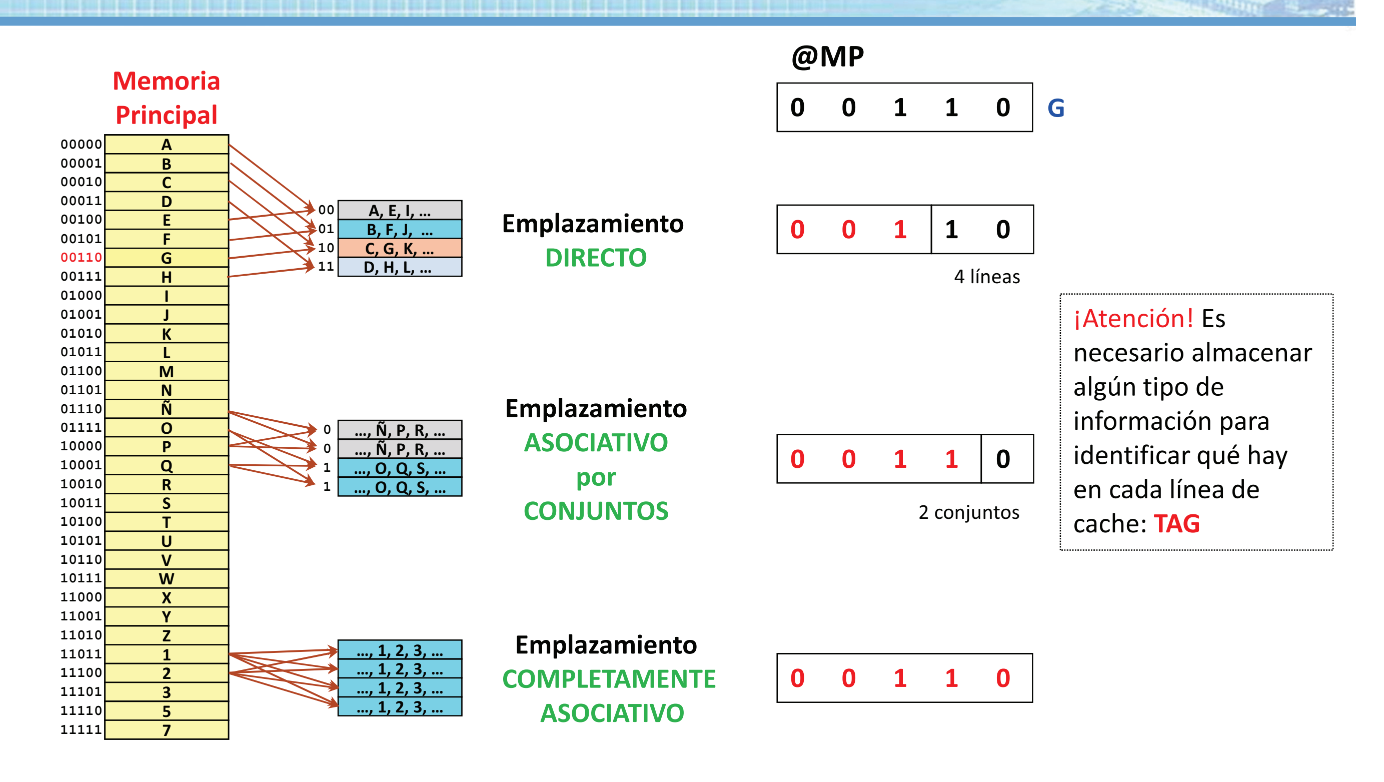


- **Objetivo:**
  - **Velocidad** de la memoria cache.
  - **Capacidad** de la memoria principal.
  - **Coste** de la memoria principal más un porcentaje razonable.
- Esta solución es posible debido a la **localidad de los programas**. La cache retiene información recientemente usada e información próxima a la recientemente usada.
- Conceptos:
  - Acierto / Fallo
  - Línea/Bloque de cache
  - Algoritmos emplazamiento
  - Algoritmos reemplazo
  - Políticas de Escritura
  - Evaluación

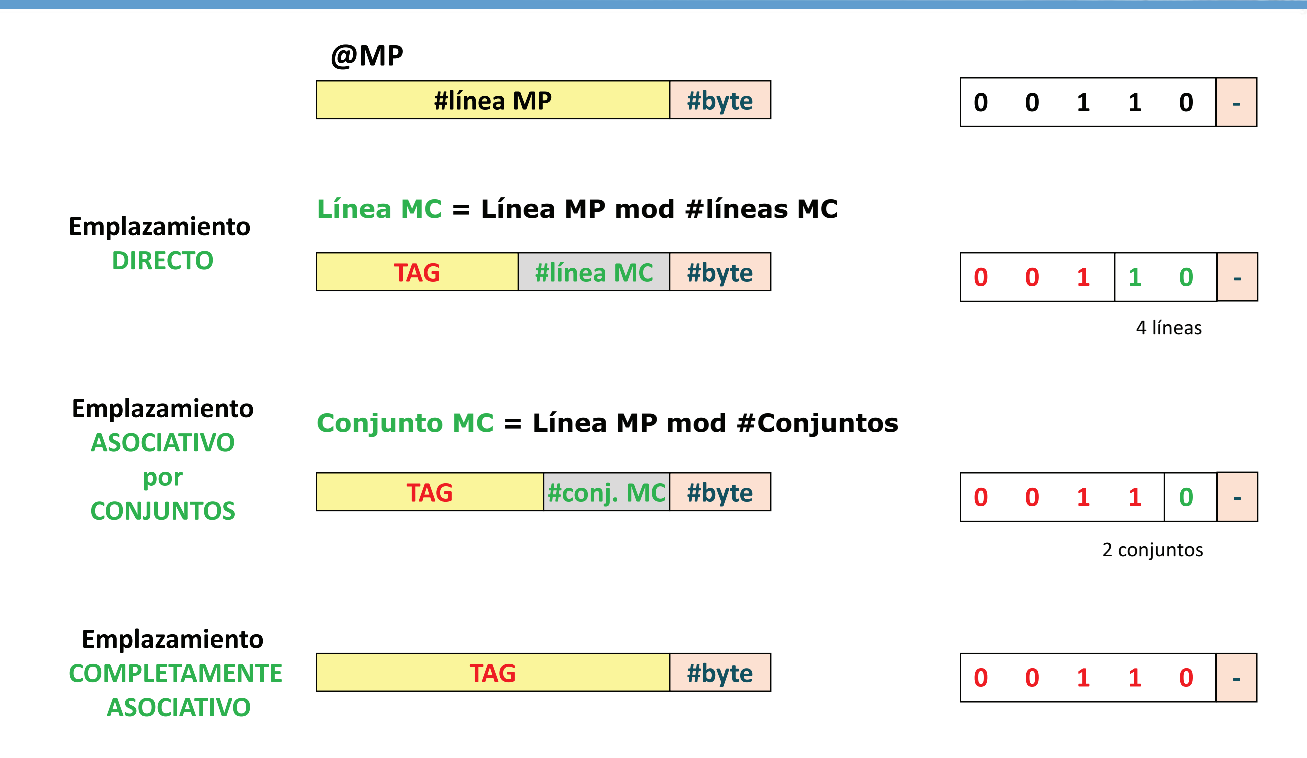
Principios de Funcionamiento de las Memorias Cache

- **En cualquier Memoria Cache hay que definir:**
  - **Algoritmo de Emplazamiento**: determina en qué bloques de MC puede colocarse una línea. Determina, también, dónde hay que buscar un dato.
  - **Algoritmo de reemplazo**: determina qué línea se ha de eliminar de la cache para dejar espacio a una nueva línea.
  - **Políticas de escritura**: determina cómo se hacen las escrituras. En cualquier caso, al final siempre se ha de escribir en MP.
- **Han de ser algoritmos hardware:**
  - Algoritmos sencillos.
  - Algoritmos rápidos.

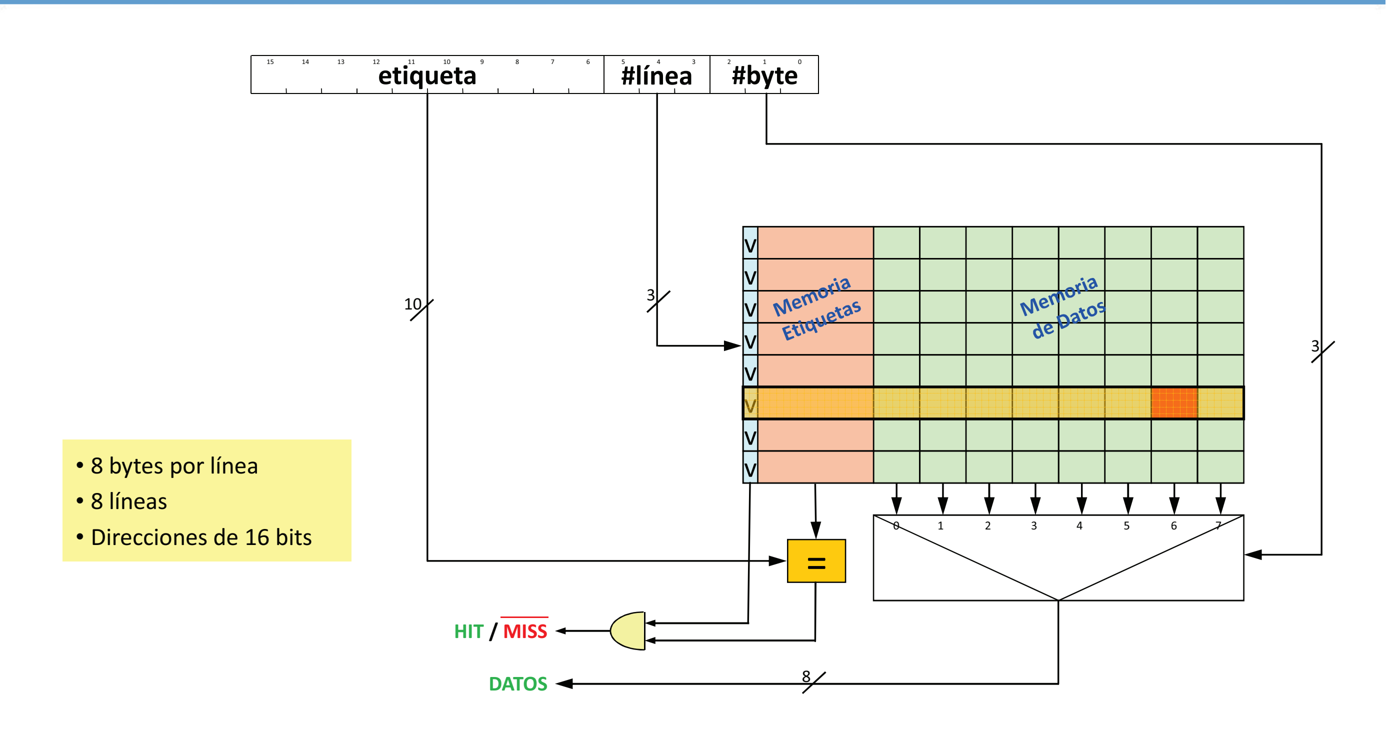
Algoritmos de Emplazamiento



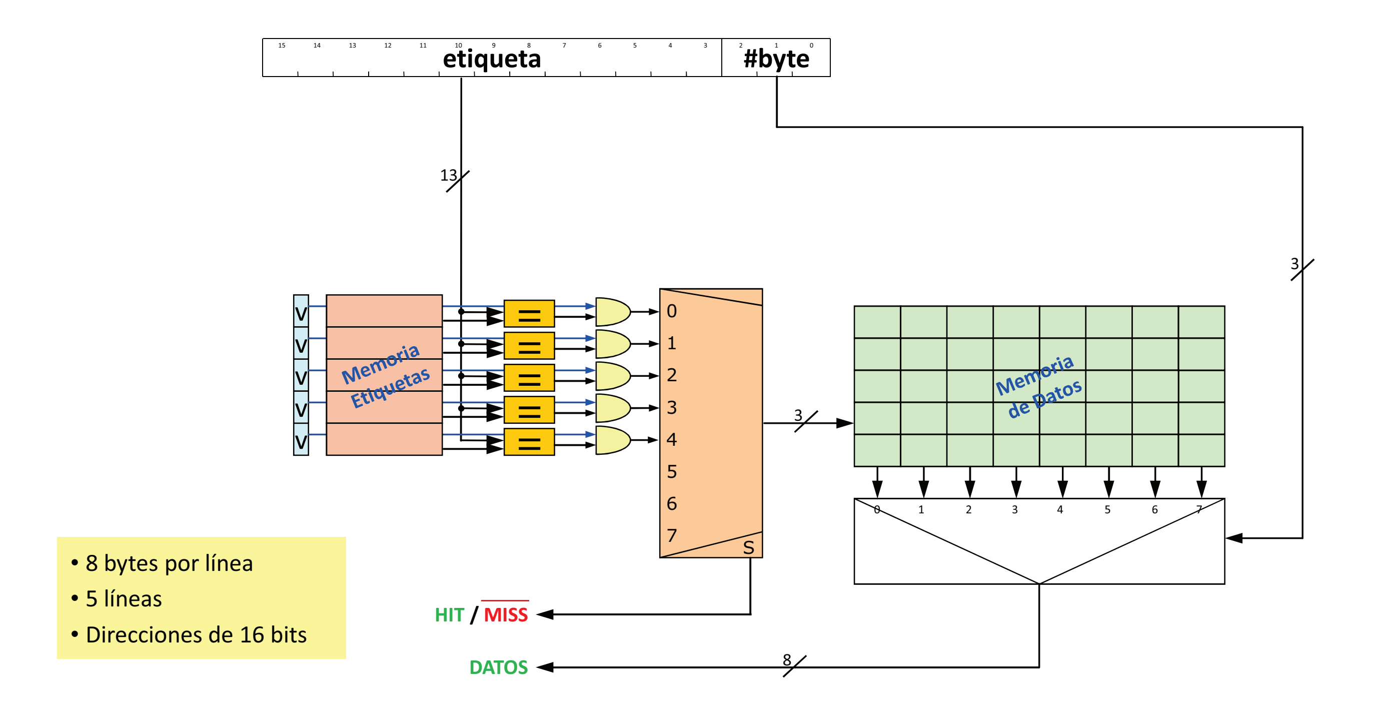
Algoritmos de Emplazamiento



Cache Directa



Cache Completamente Asociativa

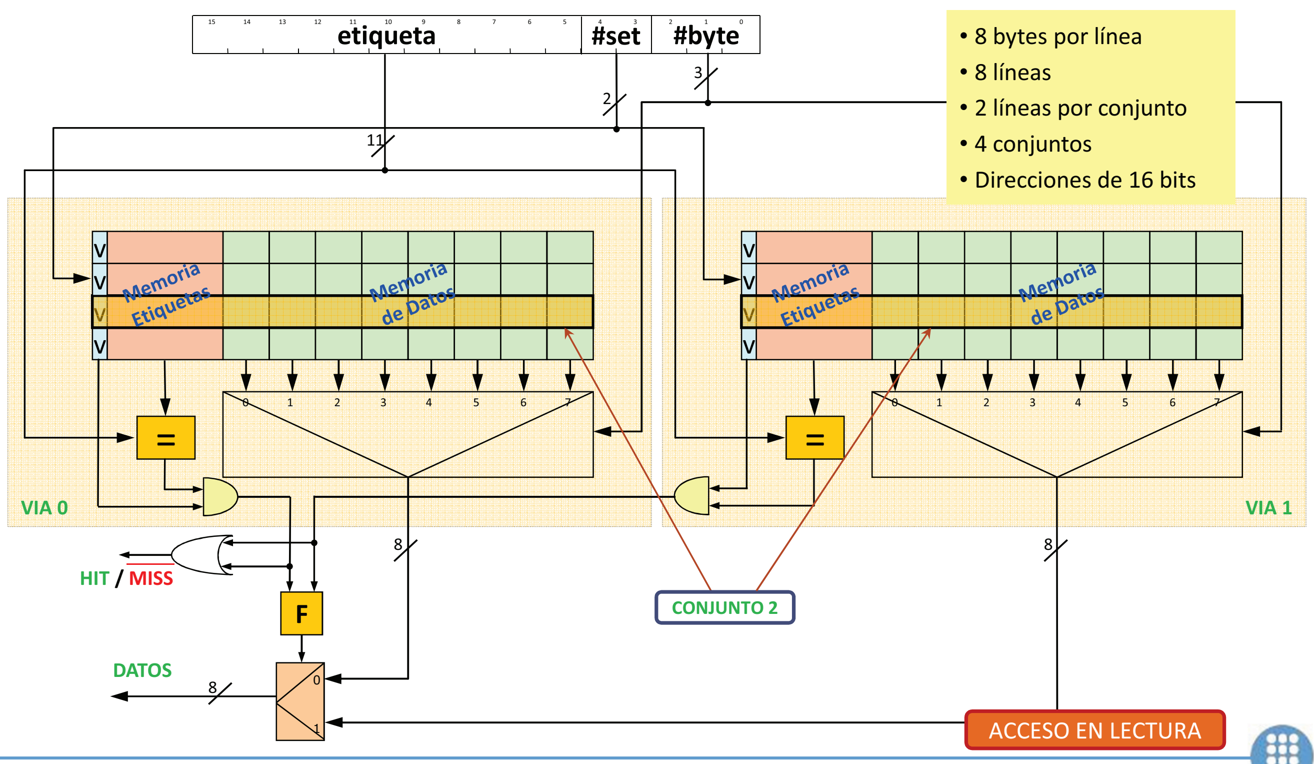


- 8 bytes por línea
- 5 líneas
- Direcciones de 16 bits

ACCESO EN LECTURA

Cache Asociativa por Conjuntos

- 8 bytes por línea
- 8 líneas
- 2 líneas por conjunto
- 4 conjuntos
- Direcciones de 16 bits



ACCESO EN LECTURA

etiqueta | #set | #byte

- 8 bytes por línea
- 12 líneas
- 3 líneas por conjunto
- 4 conjuntos
- Direcciones de 16 bits

VIA 0 · VIA 1 · VIA 2

Memoria Etiquetas · Memoria de Datos

CONJUNTO 2

HIT / MISS

DATOS

F

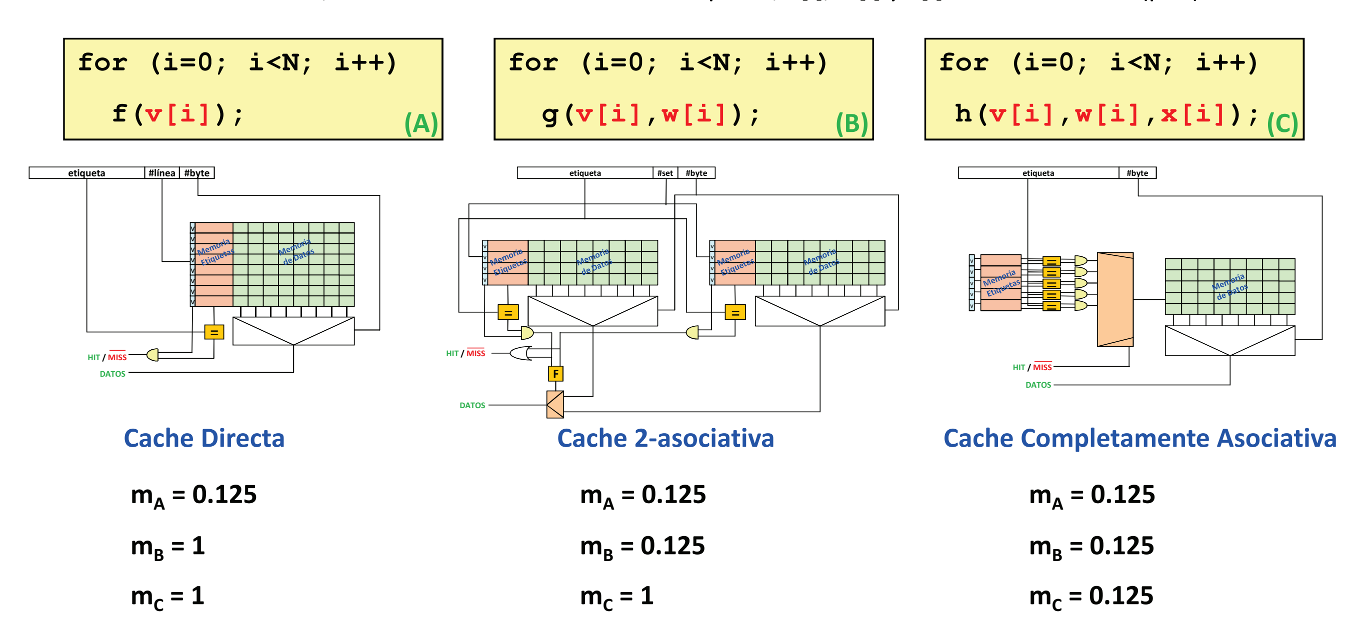
¿3-Asociativa?

ACCESO EN LECTURA

Evaluación Algoritmos emplazamiento

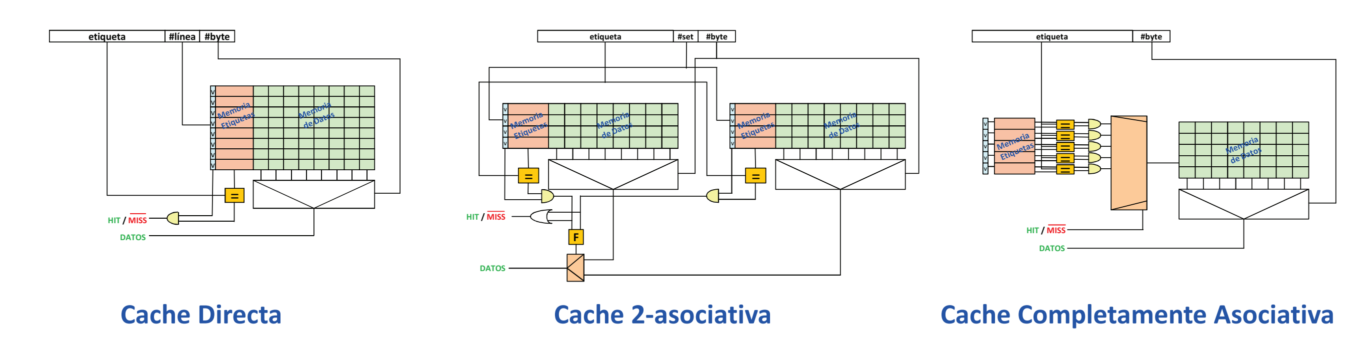
- **Tasa de fallos. Usando códigos con localidad espacial.**

Líneas de 32B; cada elemento de un vector ocupa 4B; v[i], w[i] y x[i] alineadas a 1GB (p.e.)



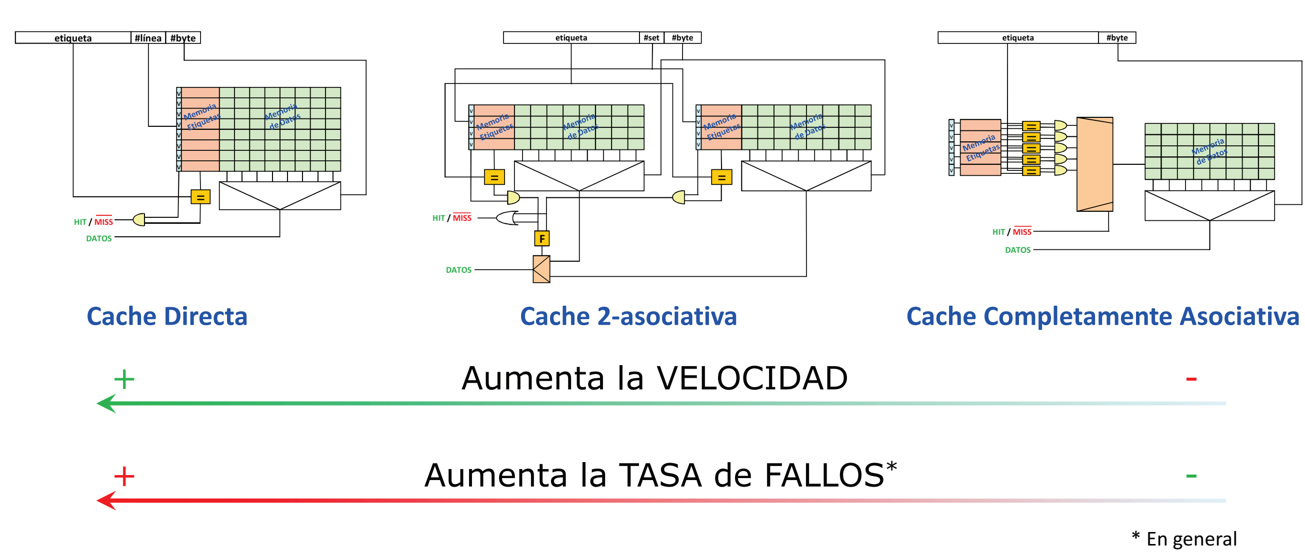
Evaluación Algoritmos emplazamiento

- **Tiempo de acceso, depende del camino crítico**



- **Cache Directa: Acceso (Memoria datos y etiquetas), comparar etiqueta y validar línea.**
- **Cache Asociativa por conjuntos: Acceso (Memoria datos y etiquetas), comparar etiqueta, validar línea y seleccionar vía.**
- **Cache Completamente Asociativa: Acceso (Memoria etiquetas), comparar etiqueta, validar línea y Acceso (Memoria datos).**

Comparación 3 algoritmos emplazamiento



- MC Directa: MC Asociativa por Conjuntos con 1 línea por conjunto.
- MC Completamente Asociativa: MC Asociativa por conjuntos con 1 único conjunto.
- ¡Atención! Para poder comparar correctamente las diversas configuraciones de cache hay que utilizar otros parámetros, como por ejemplo, el tiempo medio de acceso (Tma).

Algoritmos de Reemplazo

- **Si se produce un fallo y el conjunto donde debe ubicarse la nueva línea está lleno,**

  ¿Qué línea del conjunto hay que reemplazar?

- **¿Algoritmo importante? Un comportamiento deficiente puede provocar que reemplacemos una línea que se va a utilizar en un acceso próximo.**
- **Algoritmos hardware muy simples. Se han de ejecutar en un plazo de tiempo muy pequeño.**
- **Los algoritmos de reemplazo se aplican en caso de fallo, pero algunos de ellos necesitan actualizar cierta información después de cada acierto.**
- **En una MC Directa no tiene sentido hablar de algoritmo de reemplazo.**

Algoritmos de Reemplazo

Reemplazo Aleatorio

- Se selecciona aleatoriamente una línea de entre todas las candidatas a ser reemplazadas.
- Es un algoritmo muy sencillo de implementar.
- A pesar de su aparente falta de sentido, funciona bastante bien.

Reemplazo FIFO (First In First Out)

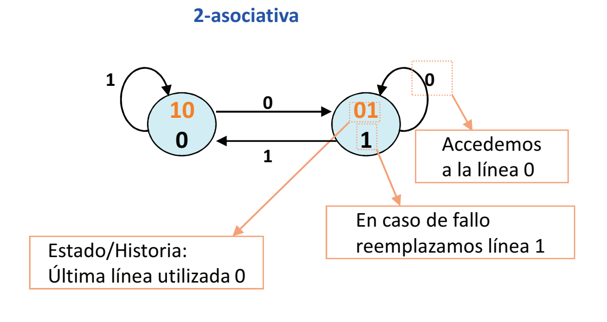
- De entre todas las líneas candidatas a ser reemplazadas, se selecciona la que lleva más tiempo en la cache.
- Es un algoritmo muy sencillo de implementar, sólo es necesario utilizar un contador con módulo.
- Tiene un comportamiento patológico no deseable porque no tiene en cuenta la utilización.

Reemplazo LRU (Least Recently Used)

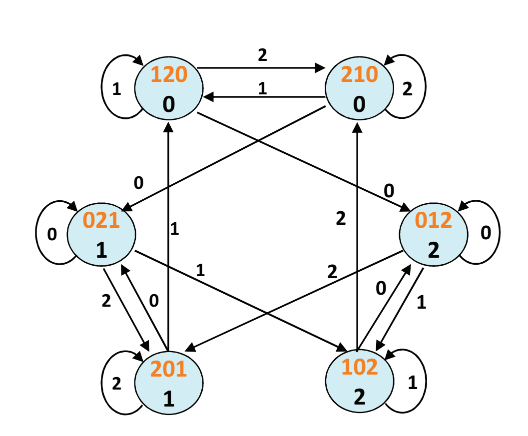
- De entre todas las líneas candidatas a ser reemplazadas, se selecciona la que lleva más tiempo en la cache sin ser utilizada.
- Este algoritmo da buenos resultados. Teniendo en cuenta el comportamiento de los programas, parece la opción más lógica.
- Sin embargo, es muy costoso de implementar si el grado de asociatividad es alto. El coste de implementar este algoritmo es n! (siendo n el grado de asociatividad).
- Normalmente se implementa un algoritmo PseudoLRU. Un algoritmo LRU ha de mantener información de en qué orden se ha accedido a todas las líneas de un conjunto. En un algoritmo pseudoLRU sólo se mantiene parte de esa información.

Algoritmos de Reemplazo

- **Posible Implementación LRU:**
  - Un registro de estado para cada conjunto. En función del estado sabemos qué línea hay que reemplazar.



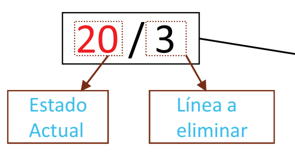
3-asociativa



Implementación imposible para un número elevado de líneas por conjunto (n).
Número de estados = n!

Algoritmos de Reemplazo

- **Posible Implementación pseudoLRU cache 5-asociativa**
  - Si tenemos n líneas por conjunto, sólo mantenemos información de las k últimas líneas utilizadas.



Estado original: 32
Accedemos en acierto a línea 1
Nuevo estado: 13
última línea accedida: 1
Penúltima línea accedida: 3

Una cache 5-asociativa necesitaría 120 estados para implementar el algoritmo LRU completo.

5-asoc	0	1	2	3	4
01 / 2	01	10	20	30	40
02 / 3	02	10	20	30	40
03 / 4	03	10	20	30	40
04 / 1	04	10	20	30	40
10 / 2	01	10	21	31	41
12 / 3	01	12	21	31	41
13 / 4	01	13	21	31	41
14 / 0	01	14	21	31	41
20 / 3	02	12	20	32	42
21 / 4	02	12	21	32	42
23 / 0	02	12	23	32	42
24 / 1	02	12	24	32	42
30 / 4	03	13	23	30	43
31 / 0	03	13	23	31	43
32 / 1	03	13	23	32	43
34 / 2	03	13	23	34	43
40 / 1	04	14	24	34	40
41 / 2	04	14	24	34	41
42 / 3	04	14	24	34	42
43 / 0	04	14	24	34	43

Algoritmos de Reemplazo

%fallos	go				tomcatv				ijpeg			
	Random	FIFO	Pseudo LRU	LRU	Random	FIFO	Pseudo LRU	LRU	Random	FIFO	Pseudo LRU	LRU
2 Kbytes	22,4	21,8	20,2	19,6	24,3	23,0	24,1	23,1	9,09	8,51	8,79	8,08
4 Kbytes	13,2	12,7	11,6	11,1	23,2	21,6	23,3	22,2	6,75	6,51	6,47	6,28
8 Kbytes	6,80	6,28	5,73	5,34	22,7	21,0	23,0	21,7	5,13	5,09	5,09	5,02
16 Kbytes	3,22	2,88	2,66	2,49	22,2	20,7	22,4	21,4	1,73	1,62	1,60	1,31
32 Kbytes	1,59	1,44	1,32	1,25	12,1	8,53	9,40	9,18	0,66	0,61	0,52	0,49
64 Kbytes	0,55	0,53	0,44	0,42	8,05	6,40	6,57	6,55	0,41	0,40	0,36	0,36
128 Kbytes	0,07	0,08	0,06	0,05	6,73	6,37	6,37	6,37	0,15	0,18	0,15	0,16

A la vista de estos resultados, se puede concluir que los algoritmos de reemplazo no influyen sustancialmente en el rendimiento de la MC.



Políticas de Escritura

Premisa: las escrituras, finalmente, se han de hacer en Memoria Principal.

¿Cuándo se actualiza la Memoria Principal?:

- WRITE THROUGH (escritura a través o escritura inmediata)
  - Se actualizan **simultáneamente** la MC y la MP.
  - El tiempo de servicio es el tiempo de acceso a MP.
  - La MP siempre está actualizada.
  - Se puede reducir el tiempo de escritura utilizando buffers.
- COPY BACK (escritura diferida)
  - En una escritura sólo se actualiza MC.
  - Para cada línea se añade un bit de control (*dirty bit*) que indica si la línea ha sido modificada o no.
  - Se **actualiza la MP** cuando la línea (modificada) ha de ser **reemplazada**.
  - Las escrituras son rápidas (velocidad de MC).
  - El tiempo de penalización en caso de fallo aumenta.
  - Durante un tiempo existe una inconsistencia entre MP y MC.

¿Qué hacer en caso de fallo en escritura?

- WRITE ALLOCATE (con migración en caso de fallo)
  - Se trae la línea de MP a MC y después se realiza la escritura.
- WRITE NO ALLOCATE (sin migración en caso de fallo)
  - La línea **NO** se trae a MC. Esto obliga a realizar la escritura directamente en MP.

Operaciones a realizar en un acceso a cache

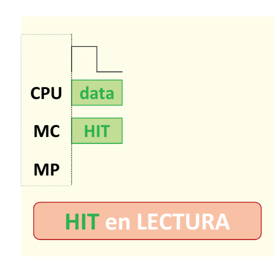
Modelo utilizado:

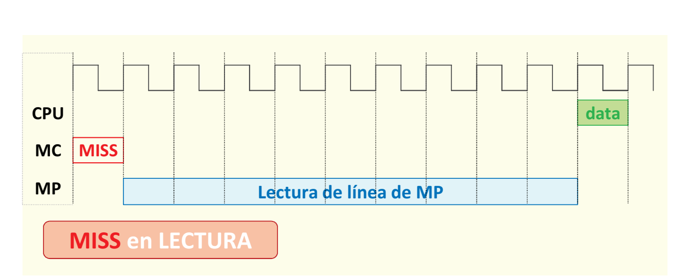
- Cache de datos
- Políticas de Escritura (2 casos):
  - Write Through + Write NO Allocate
  - Copy Back + Write Allocate
- Tamaño de línea: 32B
- Tiempo de servicio en caso de acierto: 1 ciclo
- Memoria Principal:
  - Lectura línea: 9 ciclos
  - Escritura línea: 9 ciclos
  - Escritura palabra: 6 ciclos

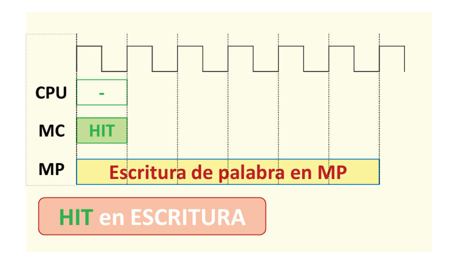


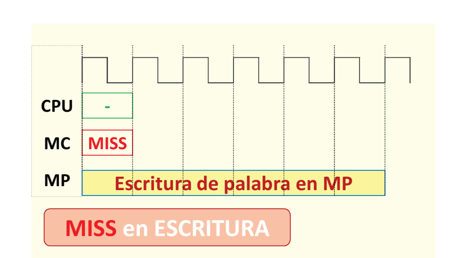
Operaciones a realizar en un acceso a cache

- **Write Through + Write NO Allocate**



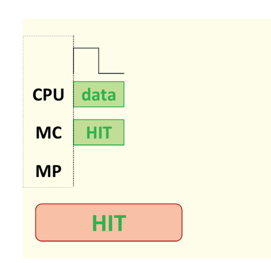


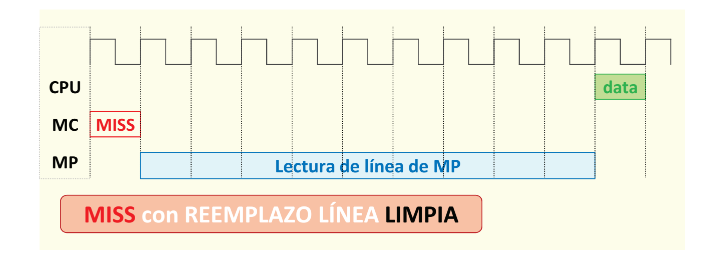


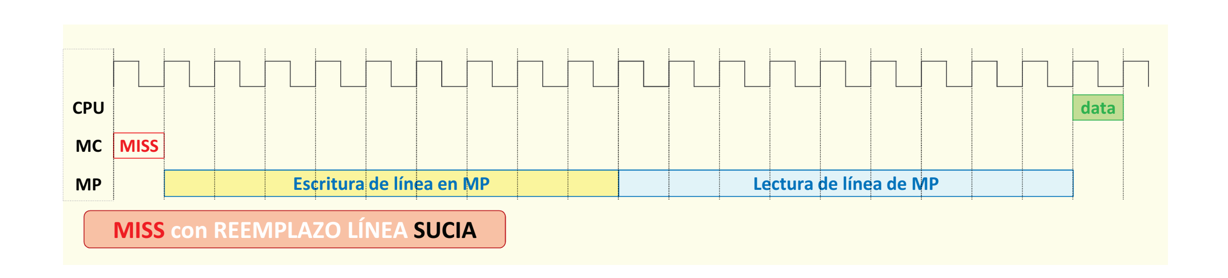


Operaciones a realizar en un acceso a cache

- **Copy Back + Write Allocate**







Influencia de los Parámetros de la Cache en el Rendimiento

- **Resultados experimentales obtenidos de la ejecución del Spec 2000:**
  - Ejecución completa, datos ref
- **Cada ejecución completa equivale $8 \cdot 10^{12}$ instrucciones simuladas en un procesador Alpha 21264**
- **Tiempos de ciclo obtenidos a partir de CACTI (¡modificados!).**
- **Datos:**
  - Número de referencias por instrucción (nr) = 0.3624 (¡real!)
  - Ciclos por Instrucción (CPI) = 1.15
  - Coste de leer una línea de MP: 100 ciclos con una memoria de 500 MHz
  - Política de Escritura: COPY BACK + WRITE ALLOCATE

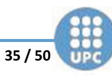
Influencia de los Parámetros de la Cache en el Rendimiento

- **Medida de Rendimiento**
  - Tasa de Aciertos

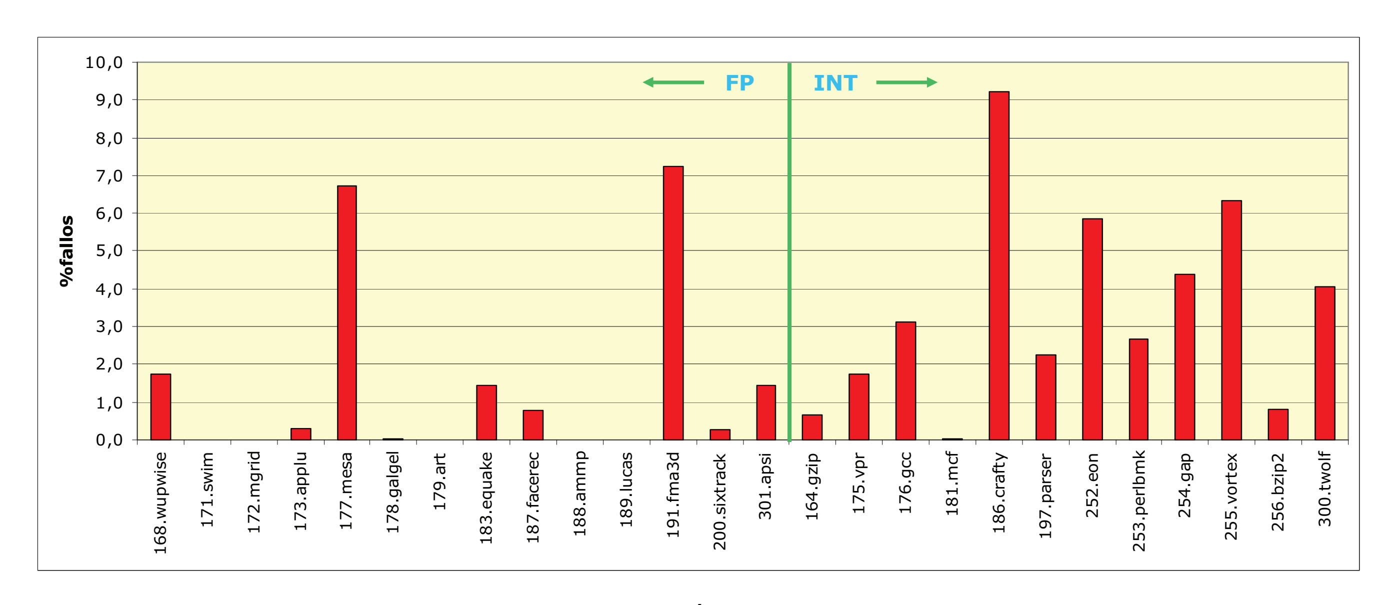
    h = #aciertos / #referencias
  - Tasa de Fallos

    m = #fallos / #referencias = 1-h

  - ¿De qué dependen?
    - ✓ Del tamaño de Cache
    - ✓ Del tamaño de Línea
    - ✓ De los algoritmos de Emplazamiento y Reemplazo
    - ✓ Del Programa evaluado



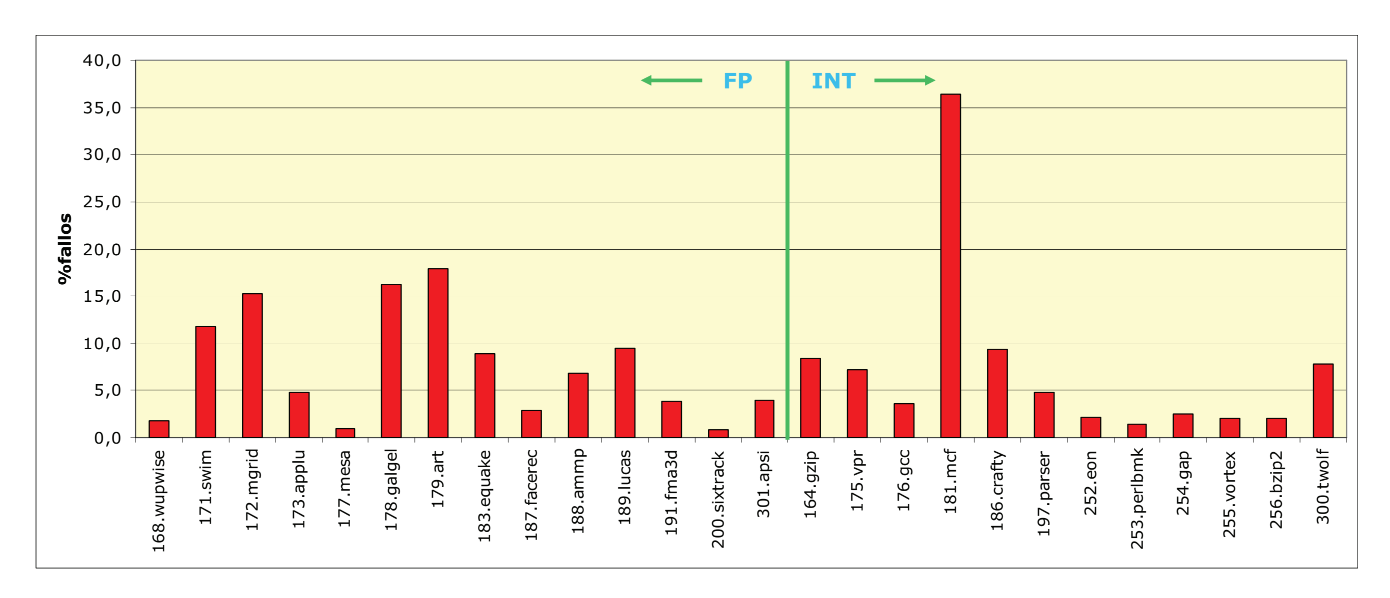
Influencia del programa en la tasa de fallos



Cache de Instrucciones directa de 4 Kbytes y líneas de 32 bytes.

¡El programa influye de forma sustancial en la tasa de fallos!

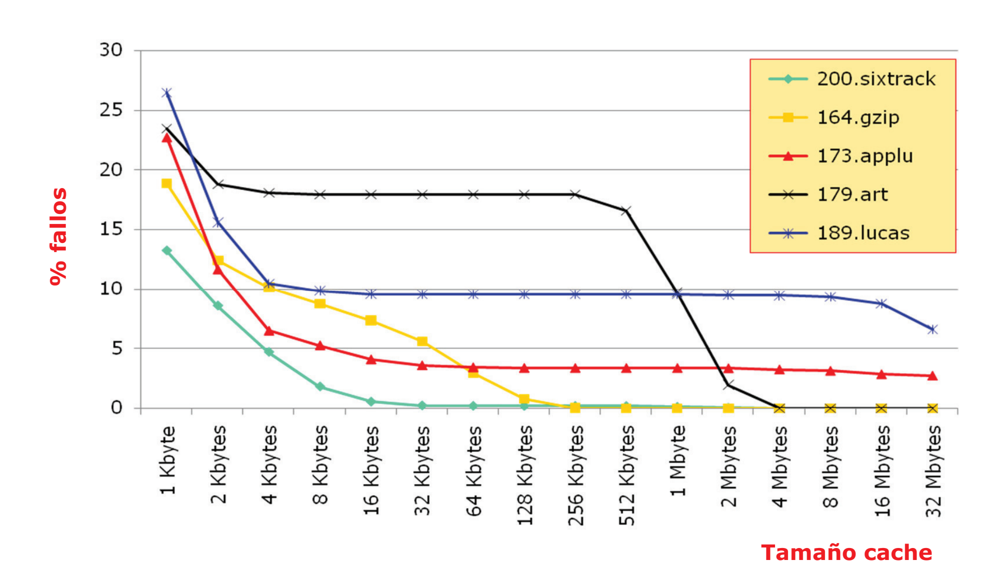
Influencia del programa en la tasa de fallos



Cache de Datos 4-asociativa de 8 Kbytes y líneas de 64 bytes.

¡El programa influye de forma sustancial en la tasa de fallos!

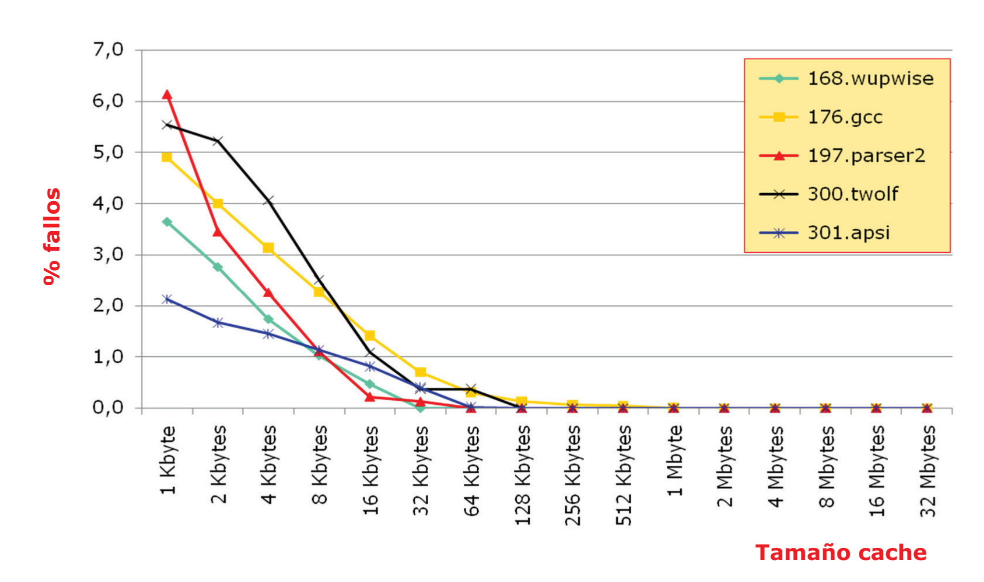
Influencia del tamaño de cache en la tasa de fallos



Cache de datos 2-asociativa con líneas de 64 bytes.

Aumentar el tamaño de cache provoca que la tasa de fallos disminuya.

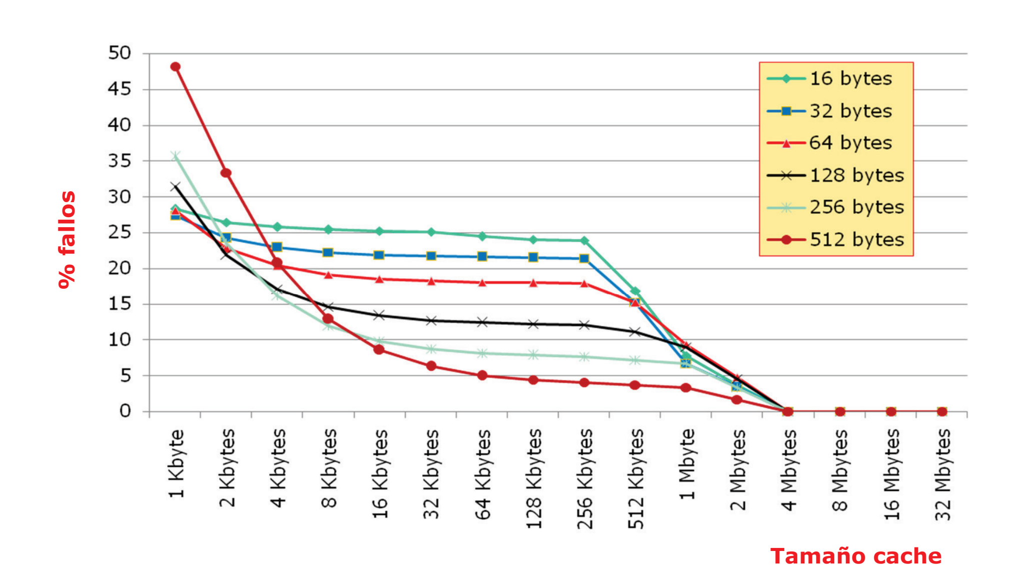
Influencia del tamaño de cache en la tasa de fallos



Cache de instrucciones directa con líneas de 32 bytes.

Aumentar el tamaño de cache provoca que la tasa de fallos disminuya.

Influencia del tamaño de línea en la tasa de fallos

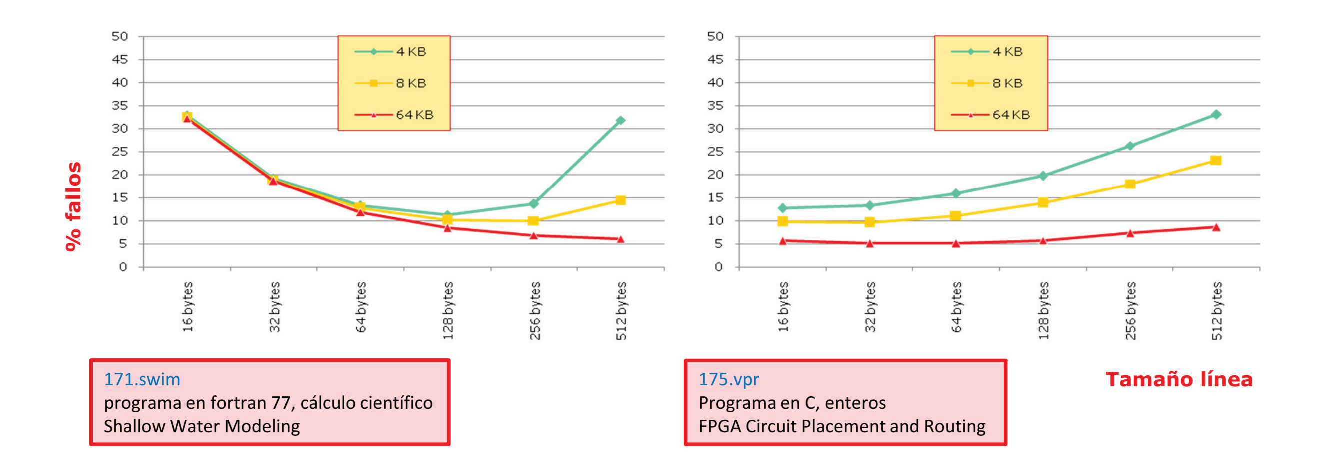


Cache de datos directa.

El tamaño de línea influye, ¡pero, en ambos sentidos!

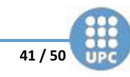
179.art
programa en C
Cálculo científico
Image Recognition

Influencia del tamaño de línea en la tasa de fallos

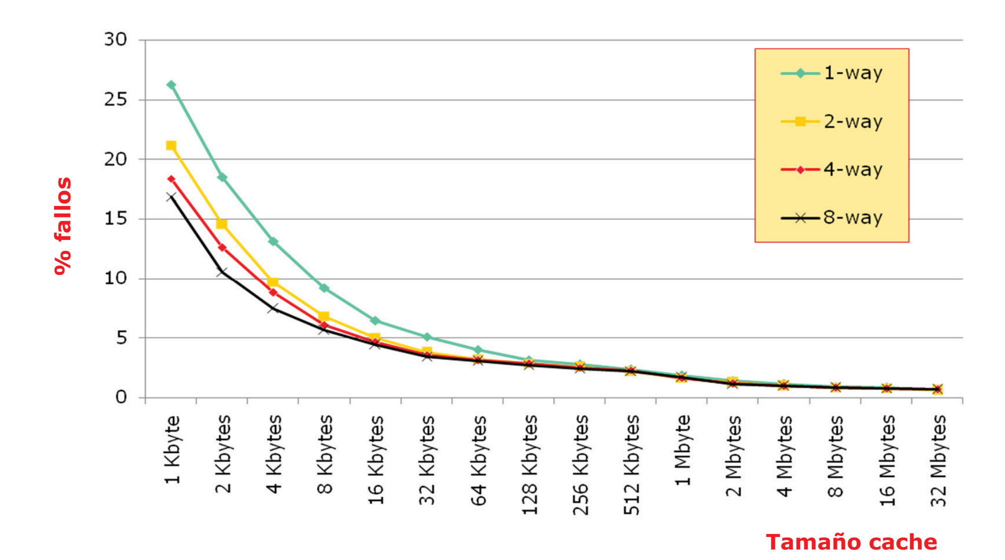


Cache de datos directa.

El tamaño de línea influye, ¡pero, en ambos sentidos!



Influencia de la Asociatividad en la tasa de fallos

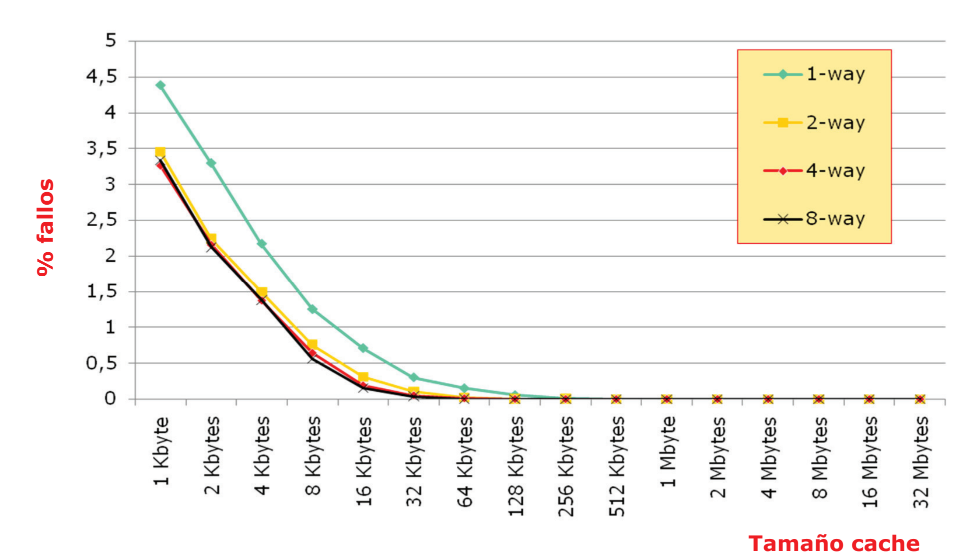


Spec 2000
26 programas
$8 \cdot 10^{12}$ instrucciones ejecutadas
$2{,}9 \cdot 10^{12}$ referencias a memoria

Cache de datos con líneas de 64 bytes.

Aumentar la asociatividad implica disminuir la tasa de fallos

Influencia de la Asociatividad en la tasa de fallos



Cache de instrucciones con líneas de 64 bytes.

Aumentar la asociatividad implica disminuir la tasa de fallos

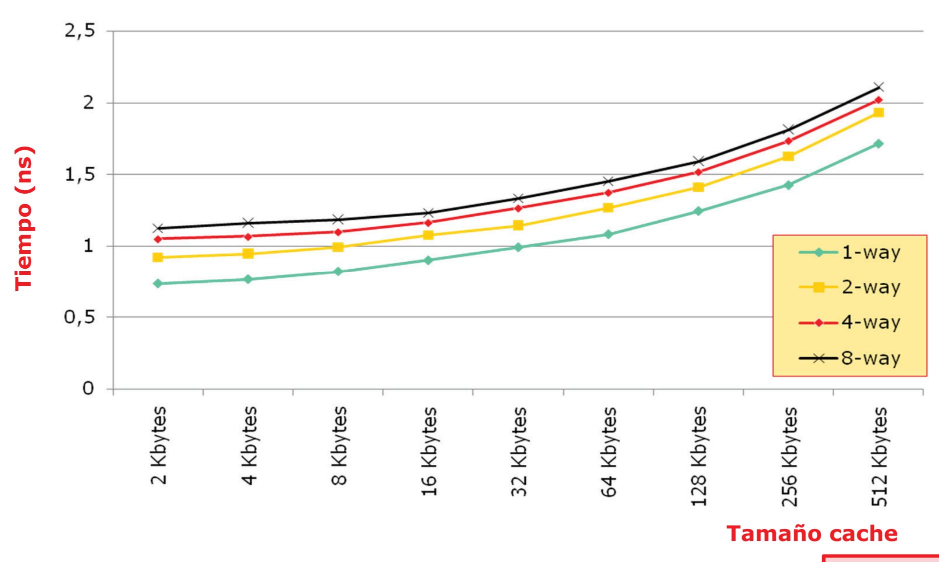
Spec 2000
26 programas
$8 \cdot 10^{12}$ instrucciones ejecutadas



Influencia de los Parámetros de la Cache en el Rendimiento

- **Medida de Rendimiento**
  - Tiempo de servicio en caso de acierto (coste de un acceso a cache en acierto)
  - ¿De qué depende?
    - ✓ Del tamaño de Cache
    - ✓ De los algoritmos de Emplazamiento y Reemplazo
  - ¿En qué influye?
    - ✓ El acceso a cache está en el camino crítico de un procesador
    - ✓ Puede comprometer el tiempo de ciclo del procesador

Influencia de los Parámetros de la Cache en el tsa



Datos de CACTI (modificado)

Cache datos con líneas de 32 bytes.

El tsa aumenta con la asociatividad y el tamaño de cache

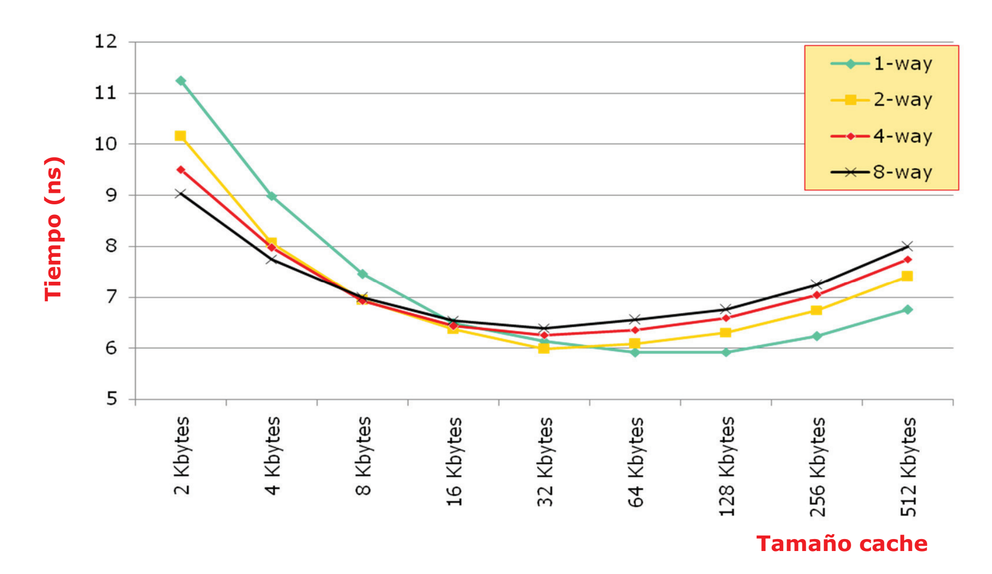


Influencia de los Parámetros de la Cache en el Rendimiento

- Medida de Rendimiento
  - Tiempo medio de acceso (coste de un acceso a memoria): Tma
  - Componentes:
    - ✓ Coste de un acceso en acierto: tsa
    - ✓ Coste de un acceso en fallo: tsf = tsa + tpf
    - ✓ Penalización de un fallo: tpf
  - Tiempo medio de acceso a Memoria: Tma = h·tsa + m·tsf = tsa + m·tpf

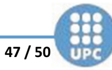
¡Atención! Esta formulación sólo es aplicable en MCs de sólo lectura porque no tiene en cuenta políticas de escritura.

Influencia de los Parámetros de la Cache en el Tma



Cache datos con líneas de 32 bytes.

Spec 2000
- Frecuencia: 500 MHz
- Coste leer línea MP: 25 ciclos



Influencia de los Parámetros de la Cache en el Rendimiento

- Medida de Rendimiento
  - Tiempo de ejecución de un programa:

    Tejec = N · CPI · Tc

    N: instrucciones ejecutadas

    CPI: ciclos por instrucción en media

    Tc: tiempo de ciclo

  ✓ CPI = CPIideal + CPImem

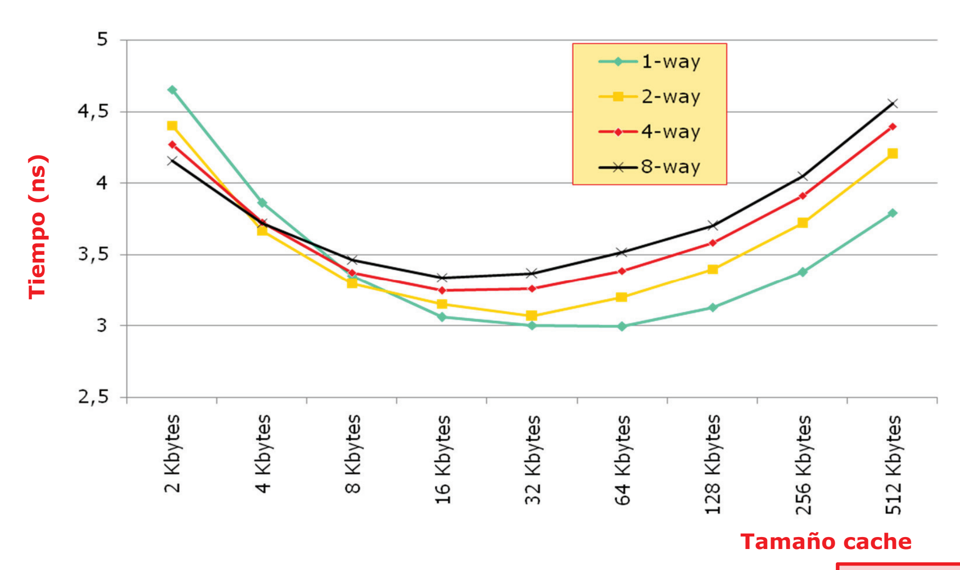
  ✓ CPImem son los ciclos que perdemos por tener una cache imperfecta ($m \neq 0$).

    CPImem = nr . (Tma – tsa)

    CPImem = nr · m · tpf (caso particular de una MC de sólo lectura)

(nr: número de referencias por instrucción)

Influencia de los Parámetros de la Cache en el Tejec



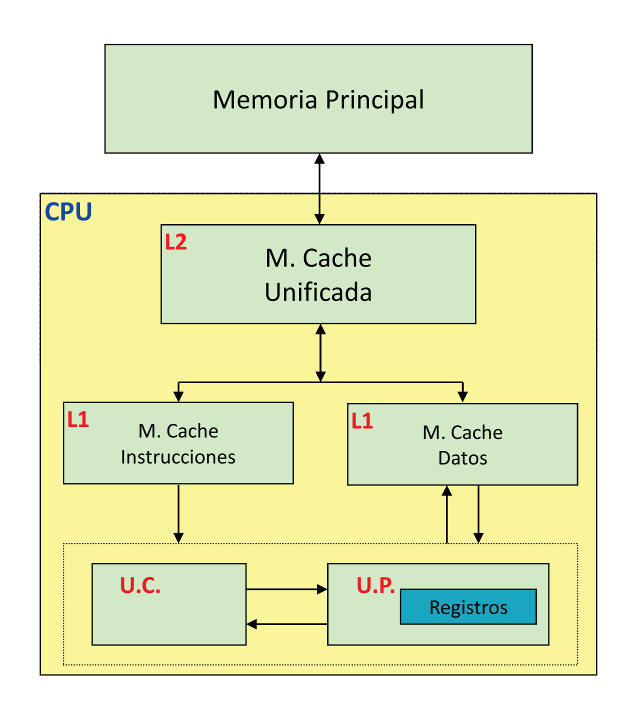
Tiempo de ejecución de 1 instrucción

Cache datos con líneas de 32 bytes.

Spec 2000
- CPIideal: 1,15
- nr: 0,3624
- Tc: tsa



Estructura Básica de un Computador Actual



Memoria Principal:
- GBytes
- Tacceso: entre 40 y 100 ciclos

L2: Memoria Cache Unificada
- MBytes
- Tacceso: entre 10 y 20 ciclos

L1: Memorias Cache Separadas
- Kbytes
- Tacceso: entre 1 y 3 ciclos

Registros:
- decenas de registros
- bancos separados: INT y FP
- Tacceso: 1 ciclo